6. NOTES

(This section contains general or explanatory information that may be helpful but is not mandatory).

6.1 Example TACO2 packet. Figure 22 is an example of a TACO2 data packet, including IP and NETBLT headers, with values shown in decimal.

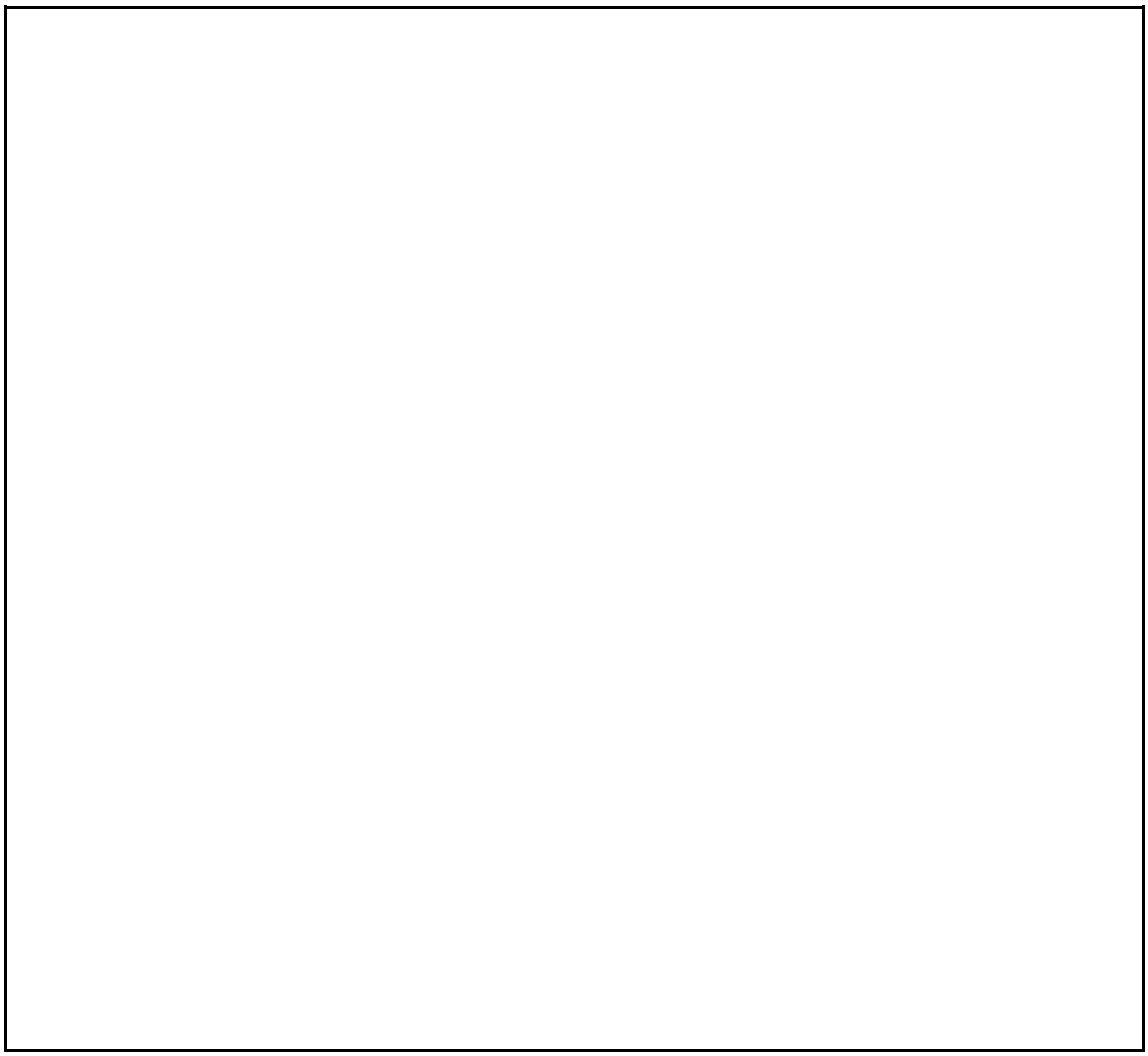

FIGURE 22. Example TACO2 packet.

The first four bits transmitted would be 0100, or decimal 4; the next four bits would be 0101, or decimal 5; that is, the values, not the field identifiers, are transmitted.

This is an internet datagram in version 4 of the internet protocol; the internet header consists of five 32 bit words, and the total length of the datagram is 70 octets, where 18 is the number of data bytes in the packet. This datagram is a complete datagram (not a fragment). Following the IP header is the NETBLT header. It indicates that this is an LDATA (type 6) packet, in version 4 of NETBLT, with 50 bytes of NETBLT header plus data. It is the first packet (packet numbers start with 0) of the first buffer, and it is both the last packet in this buffer and the last buffer in this transmission. The burst size is now seven packets, and the burst interval is seven seconds.

The actual hexadecimal values transmitted for this packet would be as follows, in octet transmission order left-to-right, top-to-bottom:

45 00 00 46 00 04 00 00 FF 1E AF 9C
80 53 08 02
81 53 02 51 D6 A7 04 06 00 32 55 4B
00 01 00 00
00 00 00 01 00 00 00 01 00 00 00 00
71 B4 00 01
00 07 1B 58 54 68 65 20 4D 49 54 52
45 20 43 6F
72 70 2E 0D 0A 1A

6.2 TACO2 NETBLT compared to RFC998 NETBLT. NETBLT as specified in Internet RFC 998 is modified in this document for use as an element of TACO2. The modifications are as follows:

a. The Transfer Size field is removed from the OPEN and RESPONSE packets.

b. The Last Buffer Touched field is added to DATA/LDATA/NULL-ACK packets. This simplifies the data timer mechanism.

c. The KEEPALIVE packet is eliminated (its function is retained, using other packet types), and packet type numbers are changed.

d. Buffer numbers start with 1.

In addition, DATA packet checksumming is mandatory, and the transfer mode must be WRITE; that is, certain NETBLT options are disallowed in TACO2. Finally, half-duplex and simplex modes of operation are defined.

6.3 SLIP drivers.

The following C language functions send and receive SLIP packets. They depend on two functions, send_char() and recv_char(), which send and receive a single character over the serial line.

```
 /* SLIP special character codes. These are OCTAL representations.
  */
 #define END             0300    /* indicates end of packet */
 #define ESC             0333    /* indicates byte stuffing */
 #define ESC_END          0334    /* ESC ESC_END means END data byte */
 #define ESC_ESC          0335    /* ESC ESC_ESC means ESC data byte */
 /* SEND_PACKET: sends a packet of length "len", starting at
  * location "p."
  */
 void send_packet(p, len)
    char *p;
    int len; {

/* send an initial END character to flush out any data that may
 * have accumulated in the receiver due to line noise
 */
   send_char(END);

/* for each byte in the packet, send the appropriate character
 * sequence
 */
    while(len--) {
        switch(*p) {
        /* if it's the same code as an END character, we send
         * a special two character code so as not to make the
         * receiver think we sent an END
         */
         case END:
            send_char(ESC);
            send_char(ESC_END);
            break;

        /* if it's the same code as an ESC character,
         * we send a special two character code so as not
         * to make the receiver think we sent an ESC
         */
         case ESC:
            send_char(ESC);
            send_char(ESC_ESC);
            break;

        /* otherwise, we just send the character
        */
        default:
            send_char(*p);
            }

         p++;
         }

    /* tell the receiver that we're done sending the packet
```

```
     */
    send_char(END);
    }

/* RECV_PACKET: receives a packet into the buffer located at "p."
 *      If more than len bytes are received, the packet will
 *      be truncated.
 *      Returns the number of bytes stored in the buffer.
 */
int recv_packet(p, len)
    char *p;
    int len; {
    char c;
    int received = 0;

    /* sit in a loop reading bytes until we put together
     * a whole packet.
     * Make sure not to copy them into the packet if we
     * run out of room.
     */
    while(1) {
        /* get a character to process
        */
        c = recv_char();

        /* handle byte stuffing if necessary
         */
        switch(c) {

        /* if it's an END character then we're done with
         * the packet
         */
        case END:
            /* a minor optimization: if there is no
             * data in the packet, ignore it. This is
             * meant to avoid bothering IP with all
             * the empty packets generated by the
             * duplicate END characters which are in
             * turn sent to try to detect line noise.
             */
            if(received)
                return received;
            else
                break;

        /* if it's the same code as an ESC character, wait
         * and get another character and then figure out
         * what to store in the packet based on that.
         */
        case ESC:
            c = recv_char();

            /* if "c" is not one of these two, then we
             * have a protocol violation.  The best bet
             * seems to be to leave the byte alone and
             * just stuff it into the packet
             */
```

```
            switch(c) {
            case ESC_END:
                c = END;
                break;
            case ESC_ESC:
                c = ESC;
                break;
                }

        /* here we fall into the default handler and let
         * it store the character for us
         */
        default:
            if(received < len)
                p[received++] = c;
            }
        }
    }
```

6.4 Notes on FEC.

6.4.1 General notes on FEC. Numerous FEC codes exist, all with different properties, with the result that the proper matching of codes to channels is an important design aspect of any system that includes FEC. FEC may be implemented in hardware, firmware, or software, or by a combination of these methods. With respect to NITFS transmissions, FEC functions shall be realized by one of the following means: as a separate hardware device, called an FEC Applique; as hardware or firmware embedded in a system; or as an integral part of a modem (or in some cases, to a modem internal to a radio); or as host-resident software. The following sections treat each of these four possibilities separately. In each case, a survey of implementation relevant to NITFS is presented.

6.4.2 Discussion of FEC appliques. An FEC Applique is an "add-on" FEC device; a separately packaged piece of equipment whose primary function is to apply FEC encoding and decoding to a data stream. FEC Appliques represent perhaps the simplest method of adding FEC to an existing SIDS system; usually only correct cabling is required, plus adjustments as needed to account for any added delays in the FEC unit.

6.4.3 Discussion of FEC-I and FEC-II. At the low-to-moderate bit rates associated with tactical communications, it is practical to implement FEC encoding and decoding in host software. The advantage of software coding will, in general, not be as great as that provided by dedicated hardware, but the portability of standardized software FEC and its relatively low cost should lead to increased interoperability. The FEC-I and FEC-II codes described in 5.4.2.1 and appendix C are designed to be implementable entirely in software on a typical workstation or portable computer.

6.4.4 Interpretation of BERT results. The BERT test described in 5.4.2.3 measures a count of successfully received frames from a given number of attempts. For an HDLC channel with random error statistics, a count value of *N* provides an estimate of the random bit-error ratio as follows:

Standard BERT test:

$$BER = 1 - (N/1000)^{1/113}$$

Short BERT test:

$$BER = 1 - (N/200)^{1/113}$$

For a SLIP channel, no exact general formula ties BERT test results to Bit Error Ratio (BER) estimates. However, the above formulae still may be used to provide approximate guidance.

6.4.5 Performance considerations. To state the performance of an FEC code, assumptions must be made in two areas. First, the noise and errors associated with the channel must be characterized. Second, a metric for data integrity is needed.

An exhaustive survey of the possibilities for these two assumptions is beyond the scope of this document. Therefore, we will evaluate several of the subject FEC codes described in the previous section using the following very general assumptions.

Channel: We assume the channel exhibits random digital errors at various levels from 10^{-5} bit error ratio (BER) to a BER of several percent.

Performance metric: We use as a performance metric the relative data throughput for 152-byte datagrams, taking into account the following four factors:

a. HDLC framing, bit-stuffing, and CRC overhead,

b. FEC redundancy,

c. Packet loss due to errors uncorrectable by the FEC,

d. Packet loss due to unrecoverable HDLC framing errors.

For example, if there is no coding overhead, and also are no errors, the relative throughput by this measure is determined solely by the overhead of the HDLC framing, bit-stuffing, and CRC.

Based on these assumptions, figure 23 compares the performance of the following five coding systems. HDLC framing is assumed in each instance.

FEC-I FEC-I as described in 5.4.1.1 of this document
FEC-II
FEC-II as described in appendix C of this document
HW-1/2
Rate 1/2 Reed-Solomon code, 6 bit symbols
HW-3/4
Rate 3/4 Reed-Solomon code, 6 bit symbols
V

Rate 1/2, constraint length 7, Viterbi Encoding
Z

No FEC

The data on figure 23 for FEC-I, HW-1/2, HW-3/4, V, and Z have been determined analytically and confirmed by experiment. The data for FEC-II are the result of analysis only.

As illustrated by figure 23, the six-bit Reed-Solomon codes (HW-1/2 and HW-3/4), and the Viterbi code (V), which have been implemented in hardware devices, outperform the software FEC specified in 5.4.2.1 and appendix C. This stems from placement of the Software FEC "above" the HDLC framing [as illustrated on figure 2, (c)], and the sensitivity of the HDLC framing mechanism to bit errors.

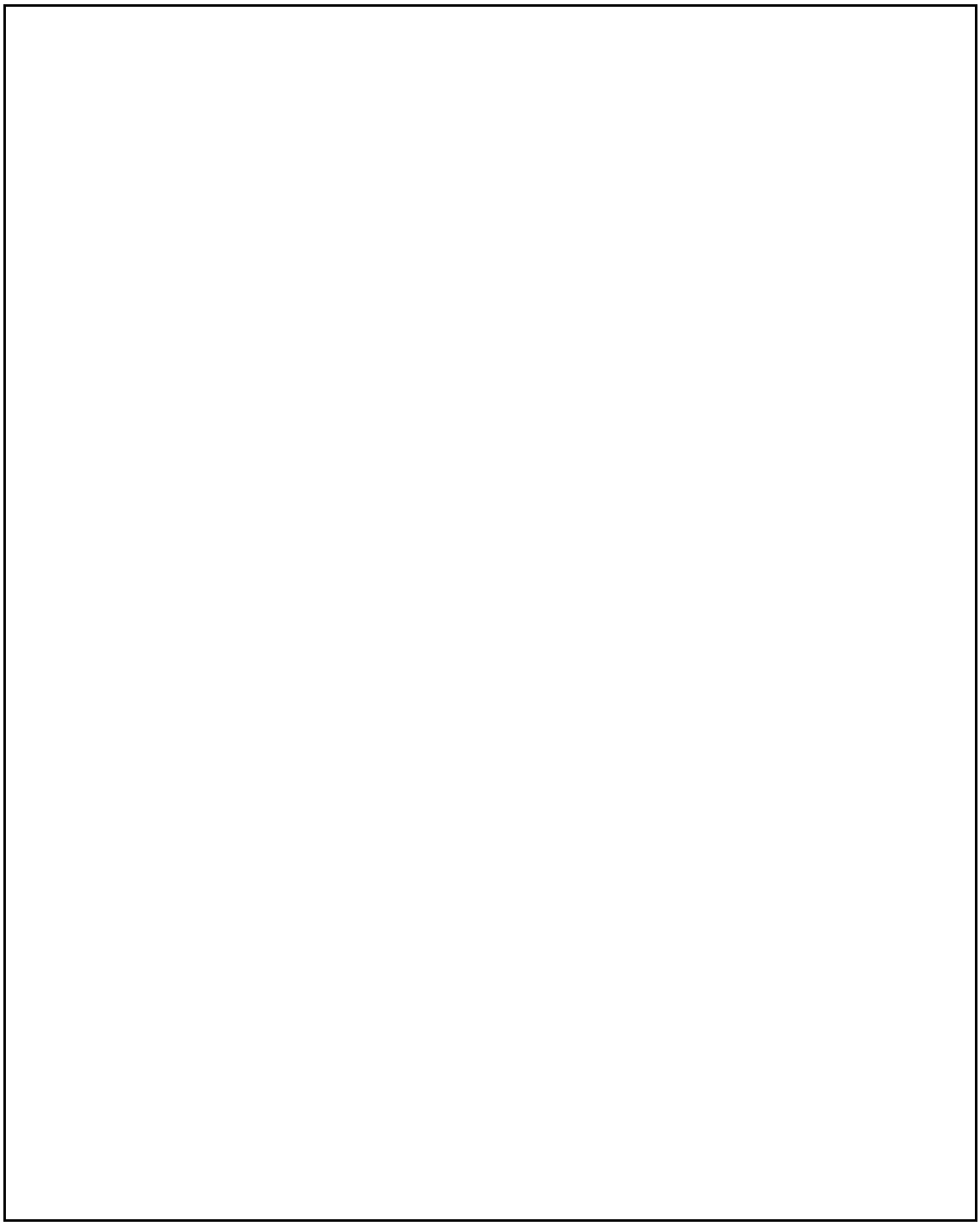

FIGURE 23. BER vs. relative throughput.

6.4.6 Selection of FEC coding options.

6.4.6.1 General discussion. Figure 23 provides some guidance in the selection of FEC coding. For random error channels, the BER may be measured by applying the BER test described in 5.4, or by other methods. Assume that the options available are no coding, FEC-I, and FEC-II. Based on this result and the data of figure 23, the no coding option is best for BER better than 5×10^{-5}, the FEC-II code preferred for BER worse than 3×10^{-3}, and the FEC-I code selected for BER between these two values.

Note that if the channel is bursty rather than random, conditions may exist in which the limited burst correction ability of FEC-I (33 bits) is exceeded, but the burst correction ability of FEC-II (790 bits) allows successful operation.

If a form of FEC is available as part of the specific datalink external to the SIDS system, the selection process becomes more complex. In many cases, the external system specific FEC will outperform the FEC-I and FEC-II codes and therefore is the first choice for routine operation of the specific SIDS system on the datalink. However, bypassing the system specific FEC and invoking FEC-I or FEC-II will allow interoperable communications between dissimilar SID systems on the same data link.

The following provides some general guidance as to the possible configurations of FEC-I and FEC-II FEC on various circuits.

6.4.6.2 Descriptions of circuits.

6.4.6.2.1 16 kbps UHF SATCOM. The circuit consists of Ultra High Frequency (UHF) transceivers communicating over satellite transponders, equipped with a KY-57 or Sunburst device operating at 16 kbps, using baseband modulation. This description also may be used for similar circuits including KY-57 encrypted line of sight (LOS) circuits using UHF or Very High Frequency (VHF) radios.

6.4.6.2.2 2.4 kbps UHF SATCOM. The circuits consist of UHF transceivers communicating over satellite transponders, equipped with a KG-84 or Sunburst device, and with internal BPSK modems operating at 2.4 kbps; or similar circuits using UHF or VHF radios operating LOS.

6.4.6.2.3 16 kbps UHF SATCOM with FEC applique. The circuit consists of UHF transceivers communicating over satellite transponders, equipped with a 16 kbps KY-57 or Sunburst device, and equipped with an FEC applique; or similar circuits using UHF or VHF radios operating LOS.

6.4.6.2.4 2.4 kbps UHF SATCOM with FEC applique. The circuits consist of UHF transceivers communicating over satellite transponders, equipped with a KG-84 or Sunburst device, and with internal Binary Phase Shift Keying (BPSK) modems operating at 2.4 kbps, and equipped with an FEC applique; or similar circuits using UHF or VHF radios operating LOS.

6.4.6.2.5 HF circuits. The circuit consists of High Frequency (HF) transceivers equipped with internal or external modems operating from 300 bps to 2400 bps.

6.4.6.2.6 HF circuits with hardware FEC. The circuit consists of HF transceivers equipped with internal or external modems operating from 300 bps to 2400 bps, with the addition of FEC either internal to the modem or by an FEC applique.

6.4.6.2.7 TRI-TAC. 16 or 32 kbps Tri-Service Tactical Communications (TRI-TAC) connection,

KY-68 encrypted.

6.4.6.2.8 Telephone circuit. Standard telephone circuit equipped with a modem that does not included retransmission-based error control such as a Bell 212/224, V.26, V.27, or V.32 modem.

6.4.6.2.9 Telephone circuit with error control. Standard telephone circuit equipped with a modem that implements retransmission protocols such as MNP4/MNP5, V.42, or PEP.

6.4.6.2.10 DAMA. Viterbi-encoded U.S. Navy Demand Assignment Multiple Access (DAMA) circuit.

6.4.6.3 Recommended modes. Table II shows the recommended configuration of FEC-I and FEC-II as a function of the above circuit descriptions. If the communication equipment includes FEC, use of FEC-I or FEC-II is not recommended.

TABLE II. Recommended modes.

Circuit Type	Recommended Modes
6.4.6.2.1	FEC-II
6.4.6.2.2	Unencoded, FEC-I, FEC-II
6.4.6.2.3	Unencoded, FEC-I, FEC-II
6.4.6.2.4	Unencoded, FEC-I, FEC-II
6.4.6.2.5	FEC-II
6.4.6.2.6	FEC-I, FEC-II
6.4.6.2.7	FEC-I
6.4.6.2.8	FEC-I, FEC-II
6.4.6.2.9	Unencoded
6.4.6.2.10	Unencoded

6.5 Effectivity summary. Some of the capabilities specified in this document are not required as of the issue date of the document. All such capabilities are marked with effectivity numbers, for example, (Effectivity 1). Each effectivity number will be replaced by a specific date in subsequent releases of this document.

6.5.1 Effectivity 1 - FEC I and Bit Error Ratio Test (BERT).

a. 4.1.6 FEC. Forward Error Correction (FEC) is a mandatory component of the TACO2 protocol stack whose use in a particular circuit is user selectable (Effectivity 1). ...

6.5.2 Effectivity 2 - FEC II.

a. APPENDIX C FEC-II CODE. (The contents of this section are (Effectivity 2) pending further implementation and testing of the proposed FEC code.)

b. 5.4.2.2.3 FEC-II. FEC-II is applied to a SLIP and/or HDLC encapsulated datalink as described in appendix C (Effectivity 2).

6.5.3 Effectivity 3 - Header abbreviation and client-controlled flow.

a. 4.1.5 Header Abbreviation sublayer. TACO2 provides a mechanism for header abbreviation across point-to-point links. Use of the header abbreviation sublayer is optional: its inclusion in any compliant implementation of TACO2 shall be mandatory (Effectivity 3).

b. 5.2.3.5 Client-controlled flow. (Effectivity 3)

c. 5.4.1 Header Abbreviation sublayer. TACO2 provides a mechanism for header abbreviation across point-to-point links. Use of the header abbreviation sublayer is optional: its inclusion in any compliant implementation of TACO2 shall be mandatory (Effectivity 3).

6.5.4 Effectivity 4 - Pull vs. push.

a. 5.1 NITFS reliable transfer server for TACO2 (TACO2 NRTS). The TACO2 NRTS described here assumes an active sender and a passive receiver ("push" operation); as of the effectivity date (Effectivity 4) the TACO2 NRTS shall also support an active receiver and passive sender ("pull" operation).

b. 5.2.9.2.5 Direction. (Effectivity 4) Until the effectivity date, operation of TACO2 is defined only for "M" set to 1; that is, TACO2 allows only active sending and passive receiving. Following that date, operation with "M" set to 0 is also permissible.

6.5.5 Effectivity 5 - Multicast.

a. 5.3.1.1 IP augmentations. ... TACO2 supports a limited form of multicasting by allowing simplex receivers to "listen in" on simplex, half-duplex, or full-duplex transmissions; (Effectivity 5: later versions of TACO2 may support acknowledged multicast).

6.5.6 Effectivity 6 - Medium Access Control layer.

a. 5.4 Data link layer. The Data Link layer in TACO2 is divided into three sublayers: Header Abbreviation, FEC, and Framing. (Effectivity 6: a Medium Access Control Layer, just below the Framing Sublayer, is under consideration.)

6.5.7 Effectivity 8 - Defense Information Systems Network (DISN).

a. DISA/JIEO Circular 9008

b. DISA/JIEO Specification 9137

c. DISA/JIEO Specification 9138

d. DISA/JIEO Specification 9139

e. DISA/JIEO Specification 9140

6.7 Subject term (key word) listing.

Error detection
Forward error correction (FEC)
Frames
HDLC
ICMP
IP
Message Transfer Facility
NETBLT
Packets
Secondary Imagery Dissemination Systems
SIDS
SLIP